2010 Third International Conference on Knowledge Discovery and Data Mining

RBACS: Rootkit Behavioral Analysis and Classification System

Desmond Lobo, Paul Watters and Xinwen Wu

Internet Commerce Security Laboratory
University of Ballarat
Ballarat, Australia
desmondlobo@students.ballarat.edu.au, {p.watters, x.wu}@ballarat.edu.au

***Abstract*- In this paper, we focus on rootkits, a special type of malicious software (malware) that operates in an obfuscated and stealthy mode to evade detection. Categorizing these rootkits will help in detecting future attacks against the business community.**

We first developed a theoretical framework for classifying rootkits. Based on our theoretical framework, we then proposed a new rootkit classification system and tested our system on a sample of rootkits that use inline function hooking. Our experimental results showed that our system could successfully categorize the sample using unsupervised clustering.

Keywords- rootkits; malware; behavioral analysis; classification; data mining

I. INTRODUCTION

This paper focuses on rootkits, which refers to software that is used to hide the presence and activity of malware from the system administrator [1]. There has been an increase of several hundred percent in both the number and complexity of rootkits over the last few years [2]. Malicious software is already a very big worldwide problem. The emergence and proliferation of rootkits are only going to serve to escalate this problem.

A large percentage of malware attacks target commercial enterprises with the intention of generating large profits for the malware's authors, while the visibility of these threats is on the decline since stealth techniques, such as rootkits, are deployed to hide the malware. This trend is clearly illustrated in figure 1. Categorizing these rootkits, we believe, will help in detecting future attacks against the business community.

The contributions of this paper are as follows. To address the limitations of the existing theoretical frameworks for classifying rootkit, we offer a revised framework. Based on this framework, we then propose a new system for categorizing rootkits and demonstrate the effectiveness of this system.

The structure of this paper is outlined below:

- Section II describes some of the previous theoretical studies that have been carried out in the field of computer virology and highlights, in particular, a recent study that was conducted at the Georgia Institute of Technology (Georgia Tech).

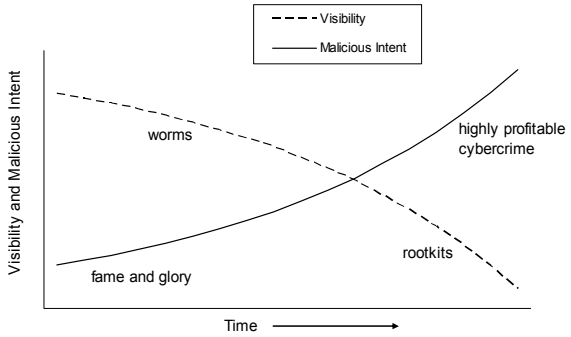

Figure 1. Visibility of Malware vs. Malicious Intent [3]

- Based on the study at Georgia Tech, we propose a revised and generalized theoretical framework for classifying rootkits in section III.
- In section IV, we explain the methodology of our Rootkit Behavioral Analysis and Classification System (RBACS).
- In section V, we test our classification system on a sample of rootkits and describe the experimental results.
- Finally, we provide a discussion of some of the related work in section VI, an outline of our future research plans in section VII, and a conclusion to the paper in section VIII.

II. PREVIOUS STUDIES IN COMPUTER VIROLOGY

Frederick Cohen was the individual that coined the phrase Computer Virus and gave the first abstract theory of computer viruses. Cohen [4] defined a computer virus as "a program that can infect other programs by modifying them to include a possibly evolved copy of itself". He went on to explain that a virus can spread throughout a network and can infect the programs of every user. Each infected program can in turn infect other programs, and so the virus spreads all through the system.

We found a recent theoretical study conducted at Georgia Tech that seems to have a similar research focus to ours. Levine et al. [5] developed a framework for classifying rootkits by making use of a previous framework for modeling Trojans and computer virus infections. Thimbleby et al. [6] categorized Trojans into four groups:

978-0-7695-3923-2/10 $26.00 © 2010 IEEE
DOI 10.1109/WKDD.2010.23

- DIRECT MASQUERADES: malicious programs that pretend to be normal programs, such as a program called dir that does NOT list a directory;
- SIMPLE MASQUERADES: programs that seem so attractive that users are tempted to use them immediately, before having them fully tested, such as a program called sex;
- SLIP MASQUERADES: programs having names that are very similar to legitimate program names, such as a program called dr that is activated if the user incorrectly types dir; and
- ENVIRONMENTAL MASQUERADES: programs that are not easily identifiable and carry out some malicious activities in addition to the tasks that the programs were intended for, such as a music CD that plays music but also executes some damaging code as a side-effect.

The work by Levine et al. [5] is more specific and restricted as they tried to classify rootkits masquerading as existing programs, either as new rootkits or as modifications of existing rootkits. We followed Levine's work to develop our framework as it is most applicable to the future trends of malware. However, we observe that Levine's framework only addresses a small class of malware. We also discover the limitations of their framework by identifying its inadequacies in classifying all types of rootkits. Hence, we propose a new generalized framework that categorizes all existing rootkits as well as future polymorphic rootkits.

Levine's study introduced set theory to categorize rootkits. More specifically, they focused on the Environmental Masquerades mentioned in Thimbleby's paper and started with the following two programs:

- p_1 was the original program, and
- p_2 was a malicious version of program p_1 that provided some additional rootkit functionality.

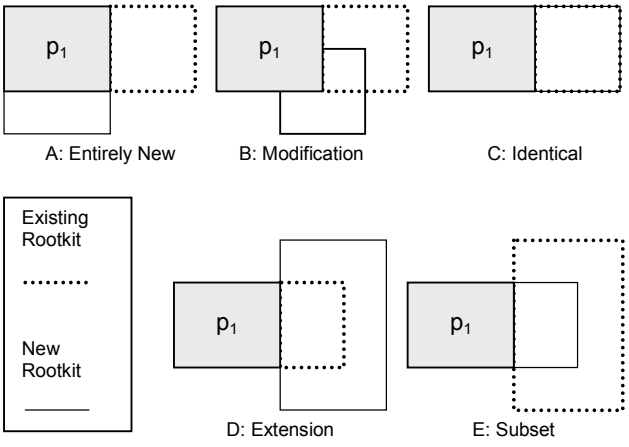

Figure 2. Theoretical Framework for Classifying Rootkits [5]

Levine and his colleagues used $\{((p_i))\}$ to symbolically represent a set of possible functionalities for a program p_i. Thus, it follows that $\{((p_1))\} \subset \{((p_2))\}$, but $\{((p_1))\} \neq \{((p_2))\}$ since there was at least one element in $\{((p_2))\}$ that did not belong in $\{((p_1))\}$. The difference Δ between p_2 and p_1 contained only those elements belonging to $\{((p_2))\}$ that were not in $\{((p_1))\}$, and this difference was defined as $\{((p'))\} = \{((p_2))\} \setminus \{((p_1))\}$.

Next, they assumed that they had identified another rootkit of p_1 called p_3 and proceeded to check the following equation:

$$\{((p_3))\} - [\{((p'))\} \cap \{((p_3))\}] = \{((p_1))\} \quad (1)$$

They could then categorize p_3 as either a brand new rootkit, a modification of an existing rootkit, or identical to an existing rootkit as follows. If equation 1 was true, Levine and his colleagues concluded that the rootkit p_3 was the same as rootkit p_2 and $\{((p_3))\}$ contained the same elements as $\{((p_2))\}$. On the other hand, if the equation was not true but some elements of $\{((p'))\}$ were contained in $\{((p_3))\}$, then p_3 could be considered a modification of p_2. If no elements of $\{((p'))\}$ were contained in $\{((p_3))\}$, then p_3 was an entirely new rootkit.

We wish to include the possibility that a newly discovered rootkit is a modification of two or more existing rootkits. It is not possible to extend the framework presented by Levine and his colleagues, however, as there is a fundamental flaw in their paper.

Figure 2 illustrates this framework for the classification of rootkits. They considered only three out of a possible five cases for classifying a newly discovered rootkit. These three cases can occur when a rootkit is entirely new (part A), a modification of an existing rootkit (part B) or identical to an existing rootkit (part C).

They failed to consider the case that a rootkit could be an extension of an existing rootkit (part D) and, more importantly, also failed to consider the possibility that the newly discovered rootkit could be a proper subset of an existing rootkit (part E). This is certainly possible: Rieck et al. [7] discovered that the Doomer worm is in fact an extension of the Gobot worm, and it is thus not inconceivable that we could find a rootkit that is an extension or subset of another rootkits.

In part E, it might be the case that some malicious hackers first release a rootkit into the wild and then, at some later date, release a "more compact" newer version of the rootkit. If this was the situation, then Levine's mathematical framework would fail to identify the newer version of the rootkit, as outlined below:

For part E, if $\{((p_3))\} \subset \{((p_2))\}$ and since $\{((p'))\} = \{((p_2))\} \setminus \{((p_1))\}$ was defined as the difference Δ between p_2 and p_1, it follows that $\{((p'))\} \cap \{((p_3))\} = \{((p_3))\} - \{((p_1))\}$. Thus, from equation 1, we have $\{((p_3))\} - [\{((p'))\} \cap \{((p_3))\}] = \{((p_3))\} - [\{((p_3))\} - \{((p_1))\}] = \{((p_1))\}$.

Since equation 1 is true, Levine and his colleagues would conclude that the new rootkit is identical to an existing

rootkit, which is not true. We propose to revise and generalize the framework in the following section.

III. GENERALIZED THEORETICAL FRAMEWORK

We now revise and generalize the theoretical framework for categorizing rootkits that was suggested by Levine and his colleagues. In their framework, they considered the possibility that a rootkit that has just been detected is actually a modification of an existing rootkit. In our framework, we wish to generalize this and include the possibility that the newly discovered rootkit has adopted some of the functionalities from two (or more) existing rootkits. This is illustrated in figure 3 and is not beyond the realm of possibility. With Rieck et al. [7] having found the coupling of two different malware families, namely the Backdoor.Zapchast and Worm.Parite families, it would certainly not be surprising to find a rootkit that had emerged from two distinct rootkit families.

For our proposed framework, we first start with the following two assumptions:

- Assume that there are n-1 known rootkits of p_1: p_2, p_3, p_4, … , p_n, and
- Assume that a new rootkit p_{n+1} has been detected.

Next, we define: $\{((X_i))\} = [\{((p_{n+1}))\}\backslash\{((p_1))\}]\cap[(\{((p_i))\}\backslash\{((p_1))\}]$ as the common functionalities of rootkit p_i (for some i, where $2 \le i \le n$) and the newly discovered rootkit p_{n+1}, and then consider the following equation:

$$[\{((X_2))\} \cup \{((X_3))\} \cup \{((X_4))\} \cup \ldots \cup \{((X_n))\}] = \Phi \quad (2)$$

If equation 2 is true, then $\{((X_i))\} = \Phi$ for all i ($2 \le i \le n$) and we can conclude that the new rootkit is completely different from each of the previously known rootkits.

If equation 2 is not true, on the other hand, then there exists some j ($2 \le j \le n$) such that $\{((X_j))\} \ne \Phi$ and we conclude that the new rootkit has some common functionality with at least one of the previously known rootkits, in this case rootkit p_j.

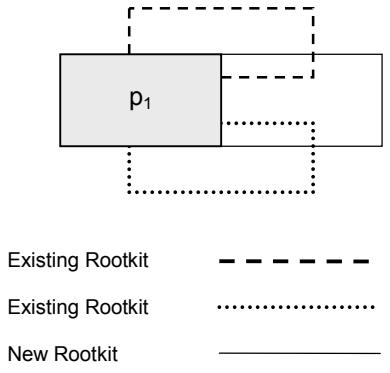

Figure 3. Generalized Framework for Classifying Rootkits

Since we are dealing with a finite number of rootkits, we can sequentially test each $\{((X_j))\}$ (for j = 2, 3, 4, … , n) to find all the rootkits that have some common functionality with the new rootkit.

A. Algorithm for Classifying Rootkit Variants

One reason for the proliferation of rootkits is the ease with which these rootkits can be implemented [8]. The developers of these rootkits often sell different versions of their product that other malicious hackers can purchase and subsequently deploy. These "toolkits" can be used to generate variants within a particular family of rootkits. Using the toolkit, even criminally minded hackers with just a minimum amount of programming experience can configure the rootkit for the particular task that they require. For instance, the hacker could specify the precise URLs that will trigger the keylogging or form grabbing capabilities of the rootkit to begin capturing data. A unique variant is generated whenever the rootkit is used for a specific purpose by different individuals or groups.

In this subsection, we describe an algorithm for classifying these rootkit variants. The algorithm makes use of our generalized theoretical framework for classifying rootkits, which will be useful in finding the functionalities that a newly discovered rootkit has in common with the existing rootkits.

For our algorithm, we will use the following notation to represent the families of rootkits:

Families of rootkits: F_1, F_2, F_3, …

Within each of these families, there will likely be several variants that have been generated using the toolkits mentioned previously, and we will use the following notation to represent these variants:

Variants within the F_i family of rootkits: p_{i2}, p_{i3}, p_{i4}, …

which are all malicious versions of program p_{i1} with some additional rootkit functionality.

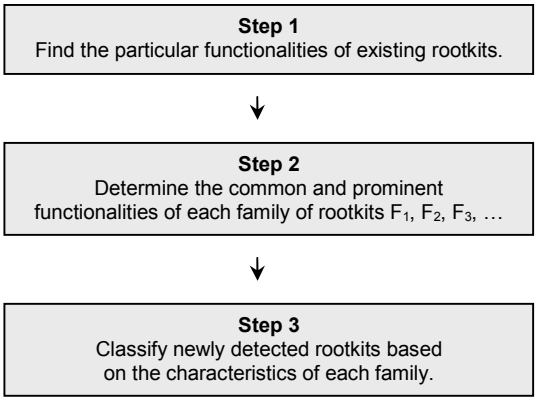

Figure 4. Algorithm for Classifying Rootkit Variants

The algorithm is illustrated in figure 4. In step 1, we first search for particular functionalities of specific rootkits. In this study, we focused on inline function hooks [12]. Rootkits create these types of hooks by overwriting the first five bytes of an API function with a JUMP instruction. (The first byte in the function is replaced with the value E9, the opcode for a JUMP instruction in assembly language, and the remaining four bytes contain a 32-bit address of some malicious code.) Thus, in this step, we keep track of all of the API functions that were found to be hooked and we also note the associated Windows processes.

Step 2 is used to determine the prominent functionalities of each family; for example, a particular family of rootkits might hook certain API functions. We make the same assumption that was made by Fu et al. [9]; that is, we assume that there would only be some slight differences between the functionality of the variants within a particular family. In other words, we use the collection of inline function hooks that were created by the rootkit to determine which family it belongs to.

In step 3, we would be in a position to use the characteristics of each of the families to classify newly detected rootkits. In order to determine if such a new threat is a variant that belongs to an existing family or possibly to an entirely new family, there are several abstraction layers that can be used to make this decision. Focusing on one of the lower layers, such as content-based signatures, might be insufficient to cope with the polymorphic techniques that are being used to prevent such matching. Common obfuscation methods that make it difficult to detect variants within a family include [10]:

- NOP-INSERTION: insert dead-code into a program;
- CODE TRANSFORMATION: rearranging the order of instructions;
- REGISTER REASSIGNMENT: replace the usage of one register with another; and
- INSTRUCTION SUBSTITUTION: swap a set of instructions with an equivalent set.

To avoid such difficulties, we suggest a higher abstraction layer. We follow the methodology that was suggested by Bailey et al. [11] and define malware by what it actually does, such as hooking certain API functions.

In section V, we describe in detail the cross-validation technique that we used to verify that a sample of rootkits was classified into appropriate families.

IV. METHODOLOGY

In this section, we describe our Rootkit Behavioral Analysis and Classification System (RBACS) and outline the process for conducting our experiment. An overview of the system is illustrated in figure 5.

Sun Microsystem's VirtualBox was installed on an Ubuntu host operating system and Microsoft's Windows XP was then installed inside the virtual environment as a guest operating system. We obtained 78 rootkit samples from the Offensive Computing website and ran each sample, one by one, in the virtual environment. After running each sample, we restored the system back to its original settings.

Rootkits make use of several different hooking techniques to remain hidden, such as import address table (IAT) hooking, export address table (EAT) hooking, inline function hooking and system service descriptor table (SSDT) hooking [12]. To test our system, we focused in this case strictly on rootkits that use inline function hooking techniques, but plan to include the other hooking techniques in our future research.

For each rootkit sample, we used McAfee's Rootkit Detective to detect the hooks had been created within the Windows XP operating system. After running this tool, a log file containing all the inline function hooks was generated. Pyroto's Parse-O-Matic parser was then used to extract the important information about each hook from the log file: the Windows process that was being affected and the API function that was being hooked.

Prior to executing the rootkits, we observed that there were a total of 26 processes running on the clean system. Furthermore, after testing all 78 rootkit samples and combining the log files that had been generated, we counted a total of 60 different API functions that had been hooked. Using a spreadsheet, we then created a large table with 78 rows (one row for each rootkit) and 1560 columns (60 columns for each of the 26 processes). The table contained 0/1 binary values, with a 1 meaning that a particular rootkit had managed to hook a certain API function in a specific Windows process. In total, the 78 rootkit samples had created 11,159 inline function hooks.

The final step in the process involved taking the binary data in the table and creating a dataset. The dataset, thus, consisted of 1560 attributes and 78 instances. We then applied the expectation-maximization (EM) algorithm on the dataset. The EM algorithm was available for implementation in the Wakaito Environment for Knowledge Analysis (WEKA) [13]. The results of our experiment are described in the next section.

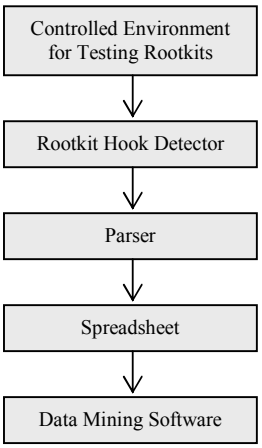

Figure 5. System Overview

TABLE I. EXPERIMENTAL RESULTS

Number of Samples	Antivirus Labels
21	ClamAV Worm.Korgo.Z **BitDefender** Backdoor.**Berbew**.Be.DAM
13	ClamAV Trojan.Crypted-29 **FProt** W32.**Berbew**.F
11	ClamAV Trojan.Crypted-29 **FProt** W32.**Berbew**.G
1	ClamAV Trojan.Qukart-8 **FProt** W32.**Berbew**.F
1	ClamAV Trojan.Qukart-10 **FProt** W32.**Berbew**.F
1	ClamAV Trojan.Qukart-17 **FProt** W32.**Berbew**.F
6	ClamAV Trojan.Crypted-28 AVGScan I-Worm.Nuwar.N **FProt** W32.**Berbew**.F
2	ClamAV Trojan.Crypted-28 BitDefender Trojan.Spy.Qukart.Z **FProt** W32.**Berbew**.G
2	ClamAV Trojan.**Bancos**-6278
14	ClamAV Worm.Feebs.AE BitDefender Win32.Worm.**Feebs**.1.Gen
1	Kaspersky Worm.Win32.**Feebs**.a
1	BitDefender Trojan.PWS.**Papras**.A
1	BitDefender Trojan.PWS.**Papras**.O
1	BitDefender Trojan.PWS.**Papras**.F
2	BitDefender Trojan.PWS.**Papras**.L

V. EXPERIMENTAL RESULTS

As mentioned, the rootkit samples for our experiment had all been obtained from the Offensive Computing website. This website uses five different antivirus scanners to label their malware samples:

FProt
BitDefender
Kaspersky
ClamAV
AVGScan

The 78 samples that we had obtained from this website had each been labeled by at least one of these antivirus scanners. For our experiment, however, we chose NOT to use these labels. We instead preferred to build a model using unsupervised clustering, where the instances had not been pre-classified. Successfully categorizing samples using unsupervised clustering would show greater evidence of the strength of our system than if we had used supervised clustering.

We chose to use the EM algorithm for clustering our dataset. An advantage of using the EM algorithm is that the most appropriate number of clusters can be found automatically. This differs from the k-means algorithm, for example, where the number of clusters needs to be specified a priori.

For each instance, the EM algorithm calculates the probability that it belongs to each of the different clusters and proceeds as follows. The number of clusters is initially set to one. The dataset is then randomly split into ten folds, with nine folds making up the training set and one fold designated as the test set. This procedure is repeated 10 times, using a different fold as a test set each time. The loglikelihood is calculated each time and the results are then averaged over all ten trials. If the loglikelihood increases, then the number of clusters is increased by one and the procedure is repeated. [13]

This cross-validation method produced the following results. The 78 samples were clustered into five families:

- The 21 samples that had been labeled as Berbew by the BitDefender Antivirus scanner were all grouped into the same family F_1.
- The 35 samples that had been labeled as Berbew by the FProt antivirus scanner were all grouped into the same family F_2.
- The two samples that had been labeled as Bancos were both grouped into the same family F_3.
- The 15 samples that had been labeled as Feebs were all grouped into the same family F_4.
- The five samples that had been labeled as Papras were all grouped into the same family F_5.

The results of our experiment are summarized in table I. We have shown that our system can be used to successfully categorize a sample of rootkits that use inline function hooking. It is important to reiterate that the sample was NOT pre-classified; we only used the labels that were assigned by the five antivirus scanners to validate our results.

Our hope is that system administrators might benefit from this system. If a system administrator detects some inline function hooks on a particular machine, he or she could then use our classification system to determine the family of that rootkit. Armed with that knowledge, the system administrator could then proceed and take further action.

VI. RELATED WORK

Kolter et al. [14] extracted n-grams from malicious executable files and, like our work, used the WEKA data mining software package to classify the malware samples into families. Their approach involved a static analysis of the executable files. Our goal, on the other hand, used a dynamic analysis approach and attempted to address a situation that a system administrator might be faced with. System administrators often find one of their machines behaving abnormally because of a rootkit infection and his or her immediate concern would be to try to determine what he or she was dealing with. Our system tries to deal with this concern.

Bayer et al. [15] had a more similar system design to ours in that they started with a dynamic analysis of each sample in a controlled environment and ended up with clusters of families. The goal of their system was to cluster malware samples based on their behavior. Our research was more

focused, however, as we used the rootkit hooks to differentiate between the various families.

VII. Future Work

This research focused on a sample of rootkits that used inline function hooking techniques. As mentioned earlier, we plan to expand our system by adding other types of rootkits and test on a larger sample set; in particular, we plan to analyze rootkits that hook the IAT, EAT and SSDT.

In addition to detecting the hooks that had been created on a system, an investigator might find other clues from a rootkit-infected machine. It might be the case, for example, that a rootkit disables the antivirus software or turns off the firewall on a system. We intend to add these and other functionalities to our dataset in the future.

When conducting our experiment, we tested just one rootkit sample at a time on a clean system, and then restored the system back to the clean state before testing the next sample. In reality, though, it is often the case that a machine is found with more than one infection. In future experiments, we would like to determine if our system can handle the situation in which more than one rootkit is running on a system.

Finally, having a large and sparse dataset, we plan to look into the possibility of using techniques from principal components analysis (PCA) for our study.

VIII. Conclusion

The first contribution of our paper was to address the limitations of the existing theoretical frameworks for classifying rootkits and to then develop a revised framework. The second contribution was to propose a new system for categorizing rootkits that was based on this framework. Our system managed to categorize 78 rootkit samples into five groups using unsupervised clustering. We then used the labels from five different antivirus scanners to verify the effectiveness of our clustering algorithm.

There still remains one unanswered question, though. Just a quick look through table 1 would be enough to conclude that there is quite a large discrepancy in the way that antivirus companies label their malware samples. The question is, why is there such a lack of consistency between antivirus labels? Bailey et al. [11] attempted to address this issue. They conjectured that some antivirus companies had the "ability to more effectively differentiate small variations in a family of malware". An antivirus company with this ability would obviously assign a greater number of unique labels to their malware samples.

From a network or system administrator's point of view, having several different labels for the same infection would certainly cause confusion. These administrators would be especially concerned during outbreaks since they could not be certain if the protection that they had in place would be adequate. Having a common label that is shared among all antivirus companies would help to alleviate this confusion and there are projects underway, such as the Common Malware Enumeration (CME) [16], to accomplish just this. Our hope is that our research would also benefit projects such as the CME.

Acknowledgment

This research was supported in part by the Westpac Banking Corporation, IBM Australia and the Victorian Government in Australia.

We would like to thank Prof Lynn Batten for her helpful comments.

References

[1] A. Emigh, "The Crimeware Landscape: Malware, Phishing, Identity Theft and Beyond", Journal of Digital Forensic Practice, Vol. 1(3), Sept. 2006, pp. 245-260

[2] McAfee, "Rootkits - Part 1 of 3: The Growing Threat", McAfee Inc., Apr. 2006

[3] OECD, "Malicious Software (Malware): A Security Threat to the Internet Economy", Organization for Economic Co-operation and Development, June 2008, OECD Ministrial Meeting on the Future of the Internet Economy

[4] F. Cohen, "Computer Viruses: Theory and Experiments", Computer and Security, Vol. 6(1), Feb. 1987, pp. 22-35, Elsevier Advanced Technology Publications

[5] J. F. Levine, J. B. Grizzard, and H. L. Owen, "Detecting and Categorizing Kernel-Level Rootkits to Aid Future Detection", IEEE Security & Privacy, Vol. 4(1), Jan. 2006, pp. 24-32

[6] H. Thimbleby, S. Anderson, and P. Cairns, "A Framework for Modelling Trojans and Computer Virus Infection", Computer Journal, Vol. 41(7), 1998, pp. 444-458, British Computer Society

[7] K. Rieck, T. Holz, C. Willems, P. Düssel1 and P. Laskov, 2008, "Learning and Classification of Malware Behavior", Detection of Intrusions and Malware, and Vulnerability Assessment, Lecture Notes in Computer Science, Volume 5137, pp. 108-125, Springer

[8] S. Hultquist, "Rootkits: the next big enterprise threat?", Information Age, Aug./Sept. 2007

[9] W. Fu, J. Pang, R. Zhao, Y. Zhang and B. Wei, "Static Detection of API-Calling Behavior from Malicious Binary Executables", Proceedings of the International Conference on Computer and Electrical Engineering, 2008, pp. 388-392, IEEE Computer Society

[10] M. Christodorescu and and S. Jha, "Static analysis of executables to detect malicious patterns", Proceedings of the 12th conference on USENIX Security Symposium, Vol. 12, 2003

[11] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian and J. Nazario, 2007, "Automated Classification and Analysis of Internet Malware", Recent Advances in Intrusion Detection, Lecture Notes in Computer Science, Vol. 4637, pp. 178-197, Springer

[12] G. Hoglund, and J. Butler, "Rootkits: Subverting the Windows Kernel", 2005, Addison-Wesley Professional

[13] I. H. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques", 2nd Edition, 2005, Morgan Kaufmann

[14] J. Z. Kolter, "Learning to Detect and Classify Malicious Executables in the Wild", The Journal of Machine Learning Research, Vol. 7, Dec. 2006, pp. 2721-2744

[15] U. Bayer, P. Milani Comparetti, C. Hlauschek, C. Kruegel and E. Kirda, "Scalable, Behavior-Based Malware Clustering", Proceedings of the 16th Annual Network & Distributed System Security Symposium, 2009

[16] D. Beck and J. Connolly, "The Common Malware Enumeration Initiative", Proceedings of the Virus Bulletin Conference, 2006